A distributed architecture for standards processing

H. Kiliccote
Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
J. H. Garrett, Jr.
Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
B. Choi
Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
K. A. Reed
Building and Fire Research Laboratory, National Institute of Standards and Technology, Washington, DC, USA

ABSTRACT: An approach to providing computer–aided support for using design standards in design systems is presented. The standards processing system we developed is composed of five major components that interact with each other using the Internet: standards processing servers which evaluate a given design to check whether it satisfies the requirements of a specific design standard; the standards processor broker which is used by the designer to identify applicable design standards; the evaluation module which manages the evaluation of a design with respect to applicable design standards; the data server which acts as a front–end between the database of the design system and the standards processing servers; and standards processing clients which display the results of evaluation to the designer and support access to the standards processor broker and standards processing servers. By separating the design system from the standards processing activities, multiple standards can be dealt with by the design system and the design system is insulated from changes in standards.

1 INTRODUCTION

In a major effort to define the nature of next generation computer aids for building design, the Carnegie Mellon University Building CAAD Consortium (CBCC) is designing and implementing a prototype software system for the preliminary design of buildings (Flemming, Coyne and Woodbury 1994).The overall architecture of SEED, which stands for Software Environment to support the Early phases in building Design, is based on a division of the preliminary design process into phases, each of which addresses a specific task. SEED intends to support each phase by an individual support module based on a shared logic and architecture. The modules envisioned for the first SEED prototype support the architectural programming, schematic layout design, and schematic configuration design.

In all three phases, various requirements defined within design standards must be identified as being applicable and addressed. Unfortunately, a very large number of design standards related to buildings are created and employed, especially in the United States. Today in the U.S., hundreds of legislative bodies, institutions, and public agencies are involved in regulating the design of civil structures and facilities. The designer must also ensure that the design conforms to consensus standards, local zoning laws, trade specifications, company standards and project specific codes.

Using such a large number of potentially applicable design standards is a tedious, laborious, and difficult task (Law and Yabuki 1992). Our objective in doing the research described in this paper is to design and implement computer–aided support for using design standards in design systems like SEED.

The first software tools to provide computer–aided support for using design standards started to appear in the early 1960's. Unfortunately each organization or software vendor started to implement their own interpretations of applicable design standards. The different problems that occur with this approach are discussed in (Garrett and Fenves 1987; Lopez, Elam and Reed 1988; Law and Yabuki 1992; Garrett and Hakim 1992; Kiliccote 1994). Briefly these problems are as follows: only some selected portions of the design standard are preselected as applicable and represented; while being represented in a computable model, design standards are usually interpreted by a software developer, who may not be very familiar with the standard; and design standards are "hard coded" in the design system. As design standards are dynamic, whenever the design standard is changed the design system must also be changed. As the design system is optimized for a specific design standard, the system cannot be easily modified to check against another design standard. The model of the standard is usable only for evaluation and cannot be manipulated for design purposes.

To overcome these problems, the idea of a generalized standards processor started to emerge in the early 1980's (Rehak and Lopez 1981; Lopez and Elam

6th International Conference on Computing in Civil and Building Engineering, July 1995, Berlin, Germany. 1995.

1984). Such a generalized standards processor was envisioned to be independent of the application and the design standard. However, the most important advantage would be that the standard writing organization would be able to develop the computational model of the standard, thus eliminating any interpretation problems introduced by an intermediate software developer.

The development of such an independent standards processor requires a separation between the design system and the standards processor and a communication protocol to control the communication between them. The SICAD (Standards Interface for Computer Aided Design) system was a software prototype developed to demonstrate the checking of designed components as described in application program databases for conformance with design standards (Lopez, Elam and Reed 1989). SICAD uses mapping functions developed by the application programmer to link to the application program database with the standards processor. However, there was no neutral data model on which SICAD could base its mappings thus making the mappings one directional and specific to the data model of the design systems. The SPEX (Standards Processing Expert) System used a standard–independent approach for sizing and proportioning structural member cross–sections (Garrett and Fenves 1987). The system reasoned with the model of a design standard to generate a set of constraints on a set of basic data items that represent the attributes of a design to be determined. The model of the standard was used by SPEX to generate constraints. These constraints were then given to a numeric optimization system to solve for an optimal set of basic data item values. SPEX assumed that the design data model and the data model used to model the standard were identical.

While both SPEX and SICAD were approaches to providing a separation between design systems and standards, they had their limitations. Thus, the more specific objective of our research is to develop a standards processor that is independent of the design system that uses it and the communication protocol necessary for communication to occur between the design system and the standards processor.

This paper briefly describes the architectural aspects of our approach to providing computer aids for modeling, accessing, and evaluating standards. Other aspects (such as functionality) of this approach are described in (Kiliccote 1994).

2 OVERALL APPROACH

Our approach in developing such an independent standards processor is to use a distributed framework to represent and reason with design standards. The standards processing system is composed of five major components that interact with each other: the *evaluation module*; the *data server*; *standards processing clients*; *standards processing servers;* and the *standards processor broker*. In this approach, shown in Figure 1, we support multiple standards processing servers that serve the needs of multiple design systems. The standards processing servers may even be running on multiple machines all over the world.

The *evaluation module*, resident within the design system (e.g., SEED), manages the evaluation of a design with respect to applicable design standards. The evaluation module uses the standards broker to identify applicable standards and then accesses the servers for those standards and requests evaluations to be performed. This module uses a standards processing client for access to the broker and servers.

The *data server* acts as a front–end between the database of the design system (e.g., the SEED database) and the standards processing servers, which use the data server to query the database to access the information related to the current design.

The *standards processor broker* is used by both the designers and the organizations that author design standards. The broker basically maintains the list of all registered design standards including their addresses on the Internet and their functionality. The organizations that author design standards register their design standards to the standards broker and the designers use the broker to determine the design standards that are applicable to the given design problem.

A *standards processing client* displays the results of evaluation to the designer and supports access to the standards processor broker and standards processing servers over the Internet.

A *standards processing server* evaluates a given design to check whether it satisfies the requirements of a specific design standard. Each standards processor may serve multiple standards processing clients.

3 PROTOTYPE

We designed and implemented a prototype of this distributed standards processor architecture using the In-

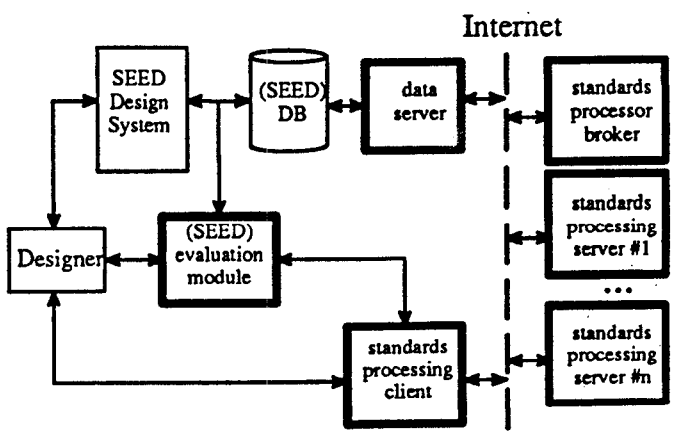

Figure 1. Overall approach for design standard support

design by triggering the CLIPS inference engine to process the converted code. Finally, the standards processor server prepares a document that contains the result of the evaluation (currently only in English, i.e., without computable information) and adds hyper–links to related documents.

In the last major step, the results are displayed to the designer, which is accomplished by the following three steps. The server returns the document to the standards processing client along with a unique identification number. The standards processing client displays the documents to the designer. Finally, the designer may request the evaluation module to perform subsequent queries using the unique identification number.

4 EXAMPLE

In this section, we provide an example of an evaluation process in the SEED environment using the prototype standards processing environment described in Section 3. Although we use the SEED system as our test–bed, the described approach is applicable to other design systems as well. For a more complete description of the design process performed in the SEED environment, see (Flemming, Coyne and Woodbury 1993). The SEED–Layout (SL) module is run by the designer and an example design is generated as shown in Figure 3.

In Figure 3, the Design Window of the SEED–Layout Module is shown. The Design Window enables the designer to construct, view and manipulate an evolving schematic layout design for a building through a combination of manual and automatic generation operations. It displays in the center part of the window a possible layout solution for the design problem. The other interface components in this figure are not relevant to our example.

The design shown is obtained by generating the three subcomponents of a One Company Headquarters Fire Station: an administration wing; an apparatus room; and a dormitory wing (shown left to right in Figure 3). Currently, fire stations are the primary building type under consideration for the development of the SEED system,

After the design is generated, the designer may choose to evaluate the design by pressing the "Evaluate" button in the Evaluation Window of the SEED–Layout Module (not shown).

In this window, using the button labeled "standards processor," the designer can choose the standards processor server that should be used to evaluate the design. This corresponds to the first step of the evaluation process described the Section 3. For now, the only standard available to choose is the Design Guide of the Department of the Army Office of the Chief Engineer DG–1110–3–145 (DAOCE 1986). The broker will eventually help to find and select among the available applicable standards. The button labeled "Evaluate" initiates an evaluation process by passing the address of the SEED data server to the standards processor (Step 2). In this step, the standards processor runs a separate CLIPS session and converts the SDL description of the Fire Station standard into the CLIPS programming language. The standards processing server accesses the design data, converts it to the CLIPS programming language and evaluates the design by triggering the CLIPS inference engine to process the converted data (Step 3). Then, the result of the evaluation appears in the standards processing client as shown in Figure 4 (Step 4).

The standards processing client that we currently use is a world–wide web client (such as Mosaic). We recognize that although using a world–wide web client to display the results of the evaluation process provides adequate support for a human to view the results,

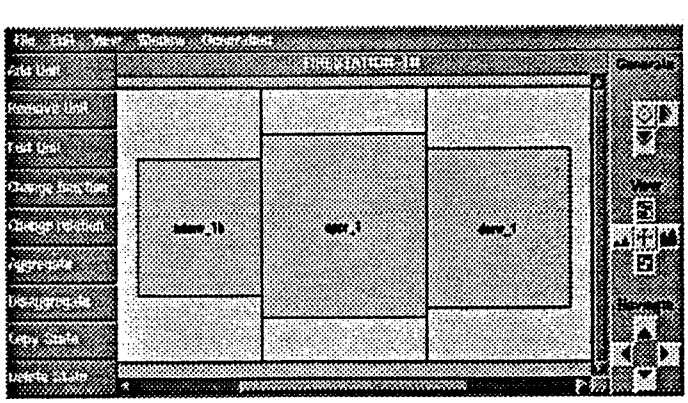

Figure 3. Design generated by SEED–Layout

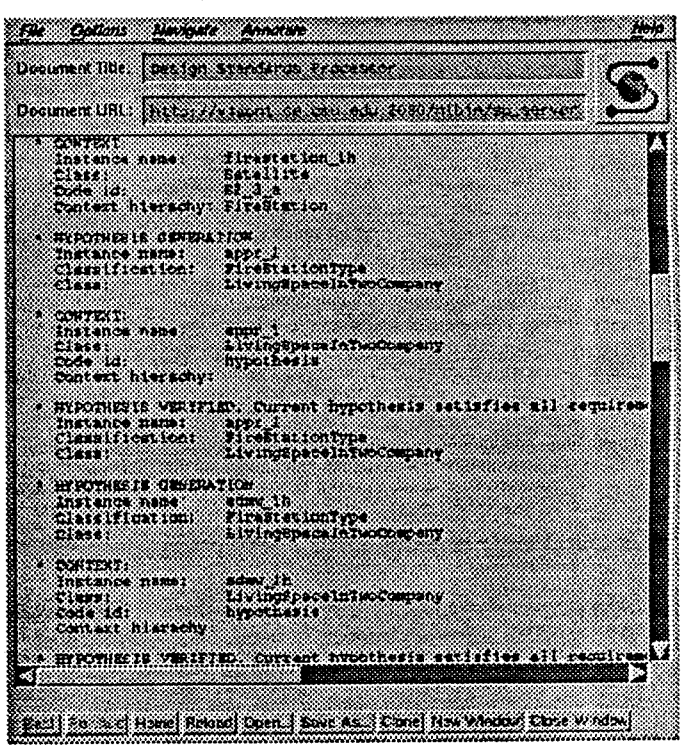

Figure 4. The result of the evaluation

design by triggering the CLIPS inference engine to process the converted code. Finally, the standards processor server prepares a document that contains the result of the evaluation (currently only in English, i.e., without computable information) and adds hyper–links to related documents.

In the last major step, the results are displayed to the designer, which is accomplished by the following three steps. The server returns the document to the standards processing client along with a unique identification number. The standards processing client displays the documents to the designer. Finally, the designer may request the evaluation module to perform subsequent queries using the unique identification number.

4 EXAMPLE

In this section, we provide an example of an evaluation process in the SEED environment using the prototype standards processing environment described in Section 3. Although we use the SEED system as our test–bed, the described approach is applicable to other design systems as well. For a more complete description of the design process performed in the SEED environment, see (Flemming, Coyne and Woodbury 1993). The SEED–Layout (SL) module is run by the designer and an example design is generated as shown in Figure 3.

In Figure 3, the Design Window of the SEED–Layout Module is shown. The Design Window enables the designer to construct, view and manipulate an evolving schematic layout design for a building through a combination of manual and automatic generation operations. It displays in the center part of the window a possible layout solution for the design problem. The other interface components in this figure are not relevant to our example.

The design shown is obtained by generating the three subcomponents of a One Company Headquarters Fire Station: an administration wing; an apparatus room; and a dormitory wing (shown left to right in Figure 3). Currently, fire stations are the primary building type under consideration for the development of the SEED system,

After the design is generated, the designer may choose to evaluate the design by pressing the "Evaluate" button in the Evaluation Window of the SEED–Layout Module (not shown).

In this window, using the button labeled "standards processor," the designer can choose the standards processor server that should be used to evaluate the design. This corresponds to the first step of the evaluation process described the Section 3. For now, the only standard available to choose is the Design Guide of the Department of the Army Office of the Chief Engineer DG–1110–3–145 (DAOCE 1986). The broker will eventually help to find and select among the available applicable standards. The button labeled "Evaluate" initiates an evaluation process by passing the address of the SEED data server to the standards processor (Step 2). In this step, the standards processor runs a separate CLIPS session and converts the SDL description of the Fire Station standard into the CLIPS programming language. The standards processing server accesses the design data, converts it to the CLIPS programming language and evaluates the design by triggering the CLIPS inference engine to process the converted data (Step 3). Then, the result of the evaluation appears in the standards processing client as shown in Figure 4 (Step 4).

The standards processing client that we currently use is a world–wide web client (such as Mosaic). We recognize that although using a world–wide web client to display the results of the evaluation process provides adequate support for a human to view the results,

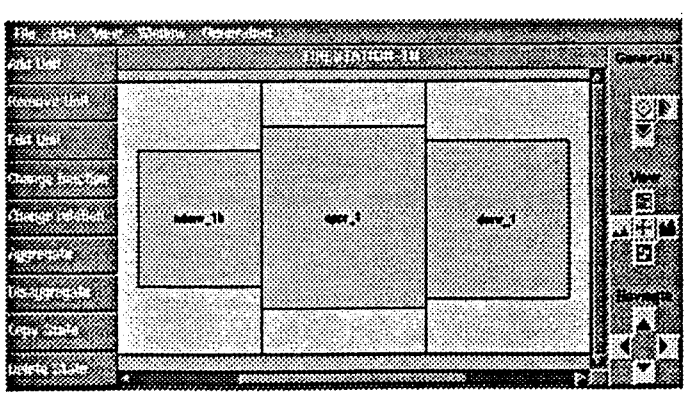

Figure 3. Design generated by SEED–Layout

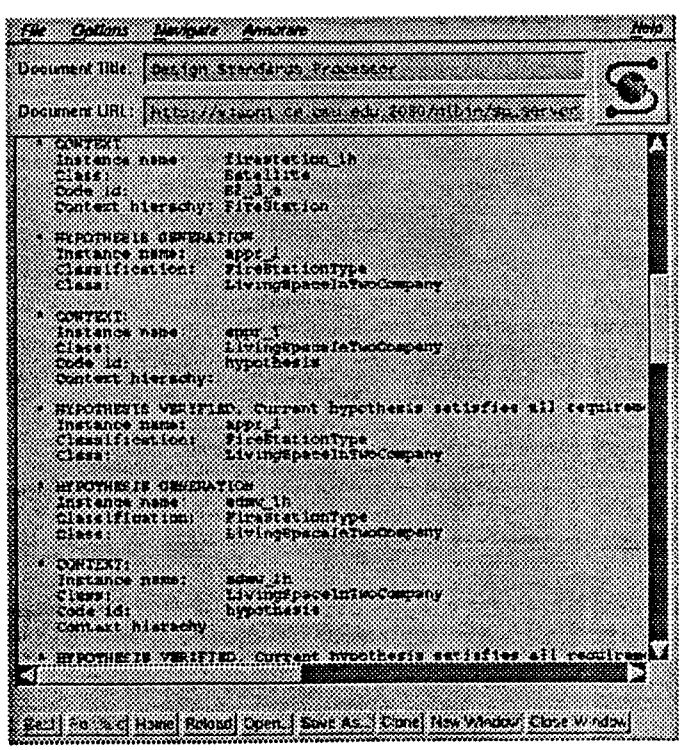

Figure 4. The result of the evaluation

extracting meaningful information from the textual representation is very difficult and other means to represent, view and query the results must be investigated in the future.

5 ADVANTAGES AND DISADVANTAGES

The distributed architecture described provides a well defined model and protocol for the communication between two applications (in this case, the SEED design system and the Fire Station standards processor server). The main advantage of this approach is that participating applications may be in physically remote locations and the type and the operating system of the machine on which they run may be different. Additionally, standards processors that use such a distributed architecture have three major advantages.

The first advantage is that, multiple design standards are easily accessed by the design system. Because a standards processor in the distributed architecture is an external (even physically remote) part of the design system, new standards processors that conform to the established communication protocol are able to be easily added. The design systems are able to access these new standards processors using the established communication protocol.

The second advantage is that, each organization is able to have its own server, which means that maintenance and legal responsibility for the design standards are distributed to the organizations that author them.

The third advantage is that, by using such a distributed approach, reference to a design standard of another organization can be made automatic and hidden from the user. Referencing a standard in this manner also ensures that the standards being referenced are always the most recent editions of the standards.

The distributed standards processing framework that we implemented was designed as a prototype to show the feasibility of this approach. From this prototype, we were able to identify some major problems related to the implementation of this distributed standards processing framework. The problems that we identified are: inadequacy of the HTTP servers; statelessness of the HTTP protocol; and inadequacy of Quanta. Each of these problems is explained in the following three paragraphs.

Inadequacy of the HTTP servers. The most important problem that we encountered is the communication overhead of the HTTP servers. We measured that on the average there is a multiple second time delay between each evaluation request and response. This delay is unacceptable in a generative system (such as SEED) that may generate tens or hundreds of alternative designs.

Statelessness of the HTTP protocol. Another problem of the current design is the statelessness of the HTTP protocol. The Router that we implemented is not a perfect solution. The basic problem is that there is no good answer to the question of how long the server must keep the separate Quanta images in memory. Currently there is a five minute access restriction. If the server is inactive for more than five minutes, then the image of Quanta is deleted from memory. This is unacceptable in a standards processor because design decisions may require hours or even days. The management problems related to using the Router are currently unanswered.

Inadequacy of Quanta. The current version of Quanta is not designed for use in a distributed system. It was designed as a compiler assuming classical interaction approaches, which means the current version of Quanta assumes that all relevant information is written to a file and supplied when Quanta is run. Because of this assumption, the information access from the data server by the Router is ad–hoc. Currently the Router requests all the information from the data server. This has at least two disadvantages: inefficiency and complexity. These two disadvantages are described in the next two paragraphs.

Inefficiency. For complex design environments, only a small portion of the design data is relevant for the evaluation of the design for a design standard. For example, the structural load information is not necessary for the evaluation of the plumbing system in a building. Thus, not passing the loading information will significantly decrease the evaluation time of the plumbing system.

Complexity. If all design information is passed to the standards processors, then the computable model of design standards would require more complex building models (i.e., at least as complex as the building models used in the design environments). For example, the building model required to model the 1990 NFiPA Standard for Dry Chemical Extinguishing Systems (NFiPA 1990) is very simple and does not require any information about, for example, foundations and retaining walls. Thus passing all the information about foundations and retaining walls for the evaluation of the fire extinguishing systems would require that the building model used in the computable model of the NFiPA Standard contain some constructs to represent foundations and retaining walls or remove such information.

6 FUTURE DIRECTIONS

To overcome the problems described in Section 5, we are working on a new distributed approach to representing and processing standards which we call the Standards Processing Framework (SPF). We are also working on a modified version of the evaluation module that provides better communication between design systems and the SPF and better control over SPF functionality. Both of these efforts are described in more detail in the following two subsections.

6.1 *Standard processing framework*

The SPF is a distributed multi–module framework for representing and processing standards. The modules in the SPF are called SPF Agents. The overall architecture proposed for the SPF is shown in Figure 5.

As shown in Figure 5, The SPF will support multiple communication protocols. In this architecture, the SPF agents may be located in the same process, in different processes (communicating using interprocess communication), on different machines in the same organization (communicating using a local network) or on different machines in different organizations distributed all over the world (communicating using the Internet).

In the SPF, unlike most other distributed systems, the criterion to be an SPF agent will be functional. In most other other systems the criterion to be an agent is behavioral (e.g., Terk 1992; Genesereth, Singh and Syed 1994). In the SPF, an entity will be an agent if and only if it can perform a specified set of functions. In SPF, agents will not be required to speak the same language or use the same communication scheme. They will not be required to understand preset semantics. Even criteria such as veracity, autonomy, and commitment will not be required, which means SPF agents may be incorrect at times (veracity), constrain another agent to perform a service even though the other agent has not advertised its willingness to accept such a request (autonomy), or prefer not to perform a service even if it has advertised its willingness to perform the service and another agent has asked it to do so (commitment).

The SPF will not impose an overall hierarchy or organization. The only criterion to be an SPF agent will be the provision of functionality that each agent must support. This functionality includes a bookkeeping facility (an agent may be questioned about the information that it supplied to or requested from other agents), a brokerage facility (an agent may be questioned about the addresses of other agents that it knows), and a context facility (an agent may be questioned about its assumptions, beliefs, language that it can speak). An agent may return incorrect or incomplete information because its assumptions or beliefs may be wrong or a communication problem may be causing a misunderstanding. The architecture by which we propose to provide this functionality is shown in Figure 6.

As shown in Figure 6, each SPF agent has three components: a *knowledge server*, a *knowledge acquirer* and a *processor*. The processor of the SPF agent is encapsulated by a knowledge acquirer and knowledge server. The knowledge server is responsible for supplying information to other agents that contact the agent. The knowledge acquirer is responsible for contacting other agents and requesting information required to perform its functions.

The knowledge server has two components: a *bookkeeper* and a *request handler*. The bookkeeper keeps track of all the information that is served to other agents. If, for any reason, a piece of information becomes invalid, the client that used that information is contacted and warned about the change. The request handler is responsible for accepting requests and sending back the results of the requests.

The knowledge acquirer has four components: a *broker*, a *bookkeeper*, an *agent discoverer*, and a *request handler*. The broker is similar to an address book. It maintains the list of all known agents and their functionality. This functionality recorded at the broker is the *context* of other agents (see below). The bookkeeper is responsible for maintaining the list of all the information that was accessed from other agents. The

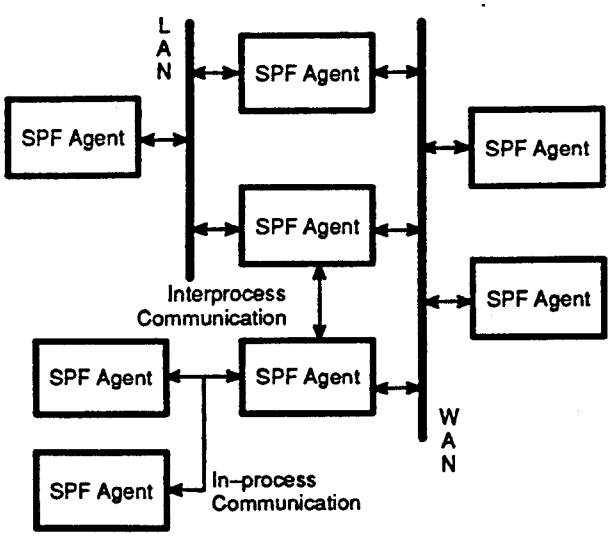

Figure 5. The Overall Architecture of the SPF

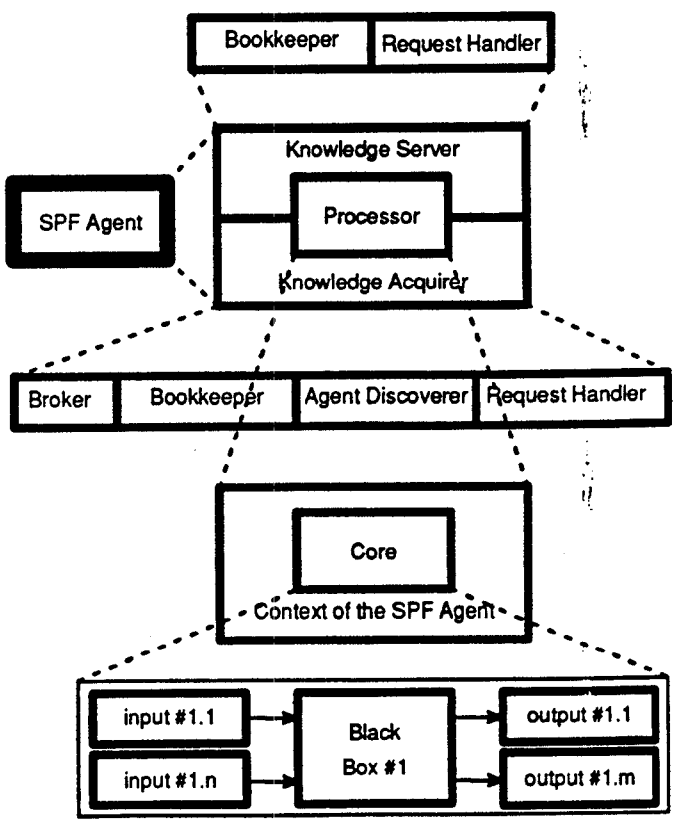

Figure 6. Anatomy of an SPF Agent

agent discoverer is responsible for finding the addresses of the agents that may fulfill an information need. The request handler is responsible for contacting other agents and requesting information.

The processor has two components: a *core* and a *context*. The core is encapsulated by the context which contains the meta information about the agent. The meta information includes the capability, assumptions, information need, and expected response time of the agent. The core is where actual computation is done. From outside the agent, it is perceived as a black box. We identified that the SPF should be able to support different representation and reasoning methodologies because the modeling of a provision may be very difficult in one representation but may be easier in another. Thus preselecting a representation and reasoning methodology may severely hinder the modeling of other provisions.

6.2 Evaluation module

Based on the SPF architecture described in the previous section, we are currently developing a new approach to support the usage of design standards during a design process. This new architecture is illustrated in Figure 7. In the previous approach (see Figure 1), the designer interacted with the design system, evaluation module and standards processing client. the evaluation module only provided support for post design checking. The role of the evaluation module in the new approach has been expanded to encapsulate the functionality of the standard processing client and the data server as well as other design standard usage functionality. The parts of the standard processing client that facilitated the communication among the designer and the standards processing agents, such as the standards processor broker and the standards processing servers, will be collected in a new component called the standards processing facilitator. This component will be responsible for correctly routing the communication between the evaluation module and the other standards processing agents.

To accomplish this new role for the evaluation module, the translation facilities that will be present in the evaluation module are being more formally defined. Again, we are using the SEED system as our test bed for this research. The data models in the SEED design system are built around a meta–level agent communication language called the Object Modeling Language (OML) (Snyder 1994). OML implements schema mapping through a shared neutral object model and language bindings. The neutral object model only exists conceptually for the applications which partially share its schema. The language bindings automatically generate code for handling low level data exchanges and state change notifications. We intend to utilize OML to map the design model in SEED to the standards model used in the standards processor agents and vice versa. This mapping is shown in Figure 8.

The design object model in the figure represents the neutral object model used by the SEED system and the standards object model represents the neutral object model used by the standards processing servers. The evaluation module stores the mapping information of the two data models and the constraints expressed over them.

In this new architecture, the evaluation process has similar steps. First the designer identifies applicable standards processing servers and contacts them through the evaluation module. The standards processing server accesses the design data and evaluates the design and the results are displayed to the designer.

We believe this approach will preserve the advantages of the distributed architecture discussed in Section 4 and also provide additional functionality such as better explanation facilities to the designer, bookkeeping facilities, and support for more active uses of design standards in the generation of designs. One immediate advantage of this new architecture is that it allows the incorporation of a wide variety of representational techniques into the same standard model, thus providing a broader, more powerful set of representations to use in modeling a design standard. Such a framework can also accommodate various types of knowledge bases and provide access to multiple design standards. Guides and heuristics supplied by designers, companies or standard organizations may be agents in this distributed framework. These agents may be very complex and be running on multiple ma-

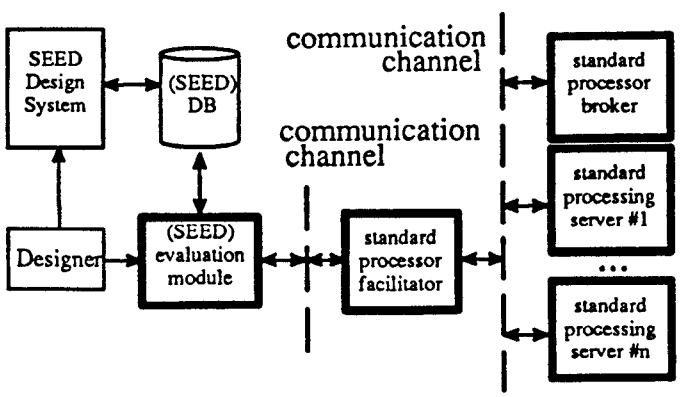

Figure 7. Proposed approach for computer–aided design standard support

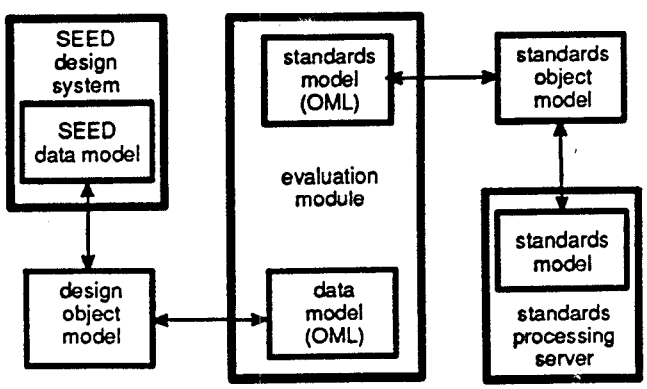

Figure 8. Object models and evaluation module

chines all over the world or be very simple and be running in the same address space on a single machine. Also because this architecture supports "true" agents, the problems that we identified in Section 5 are also solved. These agents will be able to retain their states between different evaluation processes and contact other agents to request information and services required to perform design evaluation.

7 SUMMARY

We presented two different distributed architectures to process design standards. In these architectures, multiple standards processing servers are supported independently of the design systems that use them. These standards processing processing servers may even be running on multiple machines all over the world.

The distributed architectures that we presented provide convenient approaches to create, maintain, access and evaluate design standards. The second proposed architecture is much more robust version of the first. By separating the design system from the standards processing activities, as supported by these architectures, the following benefits occur: multiple standards can be easily accessed and used in a variety of ways by the design system; the computable models of the design standards can be maintained by the organizations that create them; and the design system is insulated from changes in standards.

8 REFERENCES

Berners–Lee, T. J, R. Cailliau and J. F. Groff. 1992. The World–Wide Web. *Computer Networks and ISDN Systems*. 25:454–459.

DAOCE (Department of the Army Office of the Chief Engineers). 1986. Fire Station Design Guide No: 1110–3–145, Washington, D.C.

Flemming, U., R. Coyne, R. Woodbury. 1993. SEED: A Software Environment to Support the Early Phases in Building Design. In *Proceedings of the Fourth International Conference on Computer Aided Design in Architecture and Civil Engineering*. Barcelona, Spain, 111–122.

Garrett, J. H., Jr., and M. M. Hakim. 1992. Object–oriented model of engineering design standards. *Journal of Computing in Civil Engineering*. 6(3): 323–347.

Garrett, J. H., Jr., and S. J. Fenves. 1987. A Knowledge–based standards processor for structural component design. *Engineering with Computers*. 2(4): 219–238.

Johnson Space Center Software Technology Branch. 1993. CLIPS Reference Manual. Houston, TX.

Kiliccote H. 1994. The context–oriented model: a hybrid approach to modeling and processing design standards. Master's thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

Kiliccote, H., J. H. Garrett, Jr., T. Chmielenski, and K. Reed. 1994. The Context–Oriented Model: An Improved Modeling Approach for Representing and Processing Design Standards. In *Proceedings of the First Congress on Computing in Civil Engineering*. ASCE, Washington, D. C., 145–152.

Law, K. H., and N. Yabuki. 1992. An integrated system for design standards processing. *Proceedings of the 1992 Computer and Building Standards Workshop*. University of Montreal. Montreal, Canada, May 12–15, 1992.

Lopez, L. A., S. Elam and K. Reed. 1989. Software concept for checking engineering designs for conformance with codes and standards. *Engineering with Computers*. 5:63–78.

Lopez, L. A., and S. L. Elam. 1984. *SICAD: A Prototype Knowledge Based System for Conformance Checking and Design*. Technical Report. Civil Engineering Studies, CESL, Research Series 9. Department of Civil Engineering. University of Illinois at Urbana–Champaign. Urbana–Champaign, IL.

NFiPA 1990. National Fire Protection Association. *Dry Chemical Extinguishing Systems*. Batterymarch Park Quincy, MA

Rehak, D. R., and L. A. Lopez. 1981. *Computer Aided Engineering – Problems and Prospects*. Technical Report of Research, Civil Engineering Studies, CESL, Research Series 9. Department of Civil Engineering, University of Illinois, Urbana, IL.

Snyder, J., U. Flemming, and R. Stouffs. 1994. Sprout Object Specification Language. Unpublished Internal Document, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Terk, M. 1992. A problem–centered approach to creating design environments for facility development. Ph.D. thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

Two modules (the *standards processing client* and the *data server*) support the communication between the standards processing servers and the design system. The *evaluation module*, resident within the design system, manages the evaluation of a design with respect to applicable design standards. The evaluation module, using the standards processing client, accesses *the standards broker* to identify applicable standards and then accesses the servers for those standards and requests evaluations to be performed. The standards processing servers access the design information using the data server and evaluate the design. The results of the evaluation are displayed to the designer using the standards processing client.